uFR e-Fiscalisation manual
v1.0

Table of contents

Digital Logic Ltd.

Address: Nemanjina 57a, 12000 Pozarevac, Serbia · **Tel:** +38112541022· **VAT:** 111385444   **Reg.** 21473642
e-mail: office@d-logic.com · www.d-logic.com

About

As of version **v5.0.60** of uFCoder library, support for e-fiscalization has been introduced. Communication with **Nexgo** and **Sunmi** devices is supported by utilizing the **ReaderOpenEx()** function from our API. Developers who are working on implementation of authorization via smart cards and Generic Identity Device Specification (GIDS) specification, can utilize uFCoder API on Android devices with NFC support, without writing additional code for interaction with internal NFC and sending APDU commands. Additionally, uFR Series hardware can be used with Android devices, it is necessary to use uFCoder API to make this possible. As such - NFC support on Android device(s) is not mandatory because of uFR Series hardware.

Sending and receiving APDU commands is also supported for e-fiscalization. These are our API functions that should be used:

- APDUHexStrTransceive
- SetISO14443_4_Mode (for uFR Series readers only)
- s_block_deselect (for uFR Series readers only)

Examples of sending/receiving APDU commands can be found in section Custom APDU commands. Along with the short demonstration of fetching the token for e-fiscalisation under the Get token section.

Usage

Advanced parameters of the **ReaderOpenEx()** function from our API are used for establishing communication with the device. Currently, our API supports: Android internal NFC along with Nexgo and Sumni, uFR Online series readers, uFR series readers. Parameters for interacting with the supported devices are as following:

- ReaderOpenEx **5 0 0 0** - Android internal NFC (Also supported on Sunmi V2 Pro)
- ReaderOpenEx **5 0 1 0** - NexGo contact card side slot
- ReaderOpenEx **5 0 4 0** - NexGo PSAM1 slot
- ReaderOpenEx **5 0 5 0** - NexGo PSAM2 slot
- ReaderOpenEx **5 0 8 0** - Sunmi PSAM slot
- ReaderOpenEx **1 ip_address 85 0** - uFR Online UDP connection
- ReaderOpenEx **1 ip_address 84 0** - uFR Online TCP/IP connection
- ReaderOpen - will use FTDI to try and open the device if it's connected via OTG cable (recommended for use with uFR series readers)

ReaderOpenEx function relies on the following parameters, To determine type of communication that will be initiated, ReaderOpenEx function relies on the following parameters:
- Reader type
- Port name
- Port interface
- Additional argument.

Parameter "port name" and "additional argument", when specified as **0**, depending on platform, should instead be passed as an empty string literal.
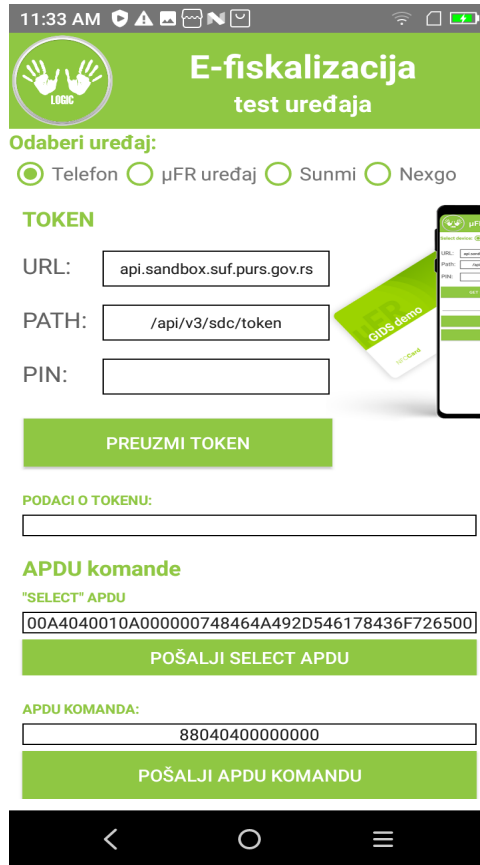
Reader Open examples

For uFR Series hardware, on Android, using the basic *ReaderOpen()* function, while the device is connected via OTG cable, will result in almost immediate port open with FTDI.
Exception to this would be uFR Online series hardware, which contains multiple methods of communication and requires usage of *ReaderOpenEx()* method, this method is also required for Sunmi/Nexgo devices.

Examples of opening the communication with the supported devices:

Android (internal NFC)

Selecting the 'Telefon' will call the following method:

uFCoder.ReaderOpenEx(5, "", 0, "");

Video demonstration of using Android device with internal NFC reader: https://youtu.be/b2QofrpRwHo

NexGo (contact card side slot)

Selecting 'Nexgo' by default will have these parameters as input, clicking on 'Reader Open' with this input will be calling the ReaderOpenEx() method as:

uFCoder.ReaderOpenEx(5, "", 8, "");

Video demonstration of using Nexgo reader can be found here: https://youtu.be/34rpoPHeHCA

Sunmi (PSAM slot)

Selecting 'Sunmi' by default will have these parameters as input, clicking on 'Reader Open' with this input will be calling the ReaderOpenEx() method as:

uFCoder.ReaderOpenEx(5, "", 8, "");

Video demonstration of using Sunmi V2 Pro reader can be found here: https://youtu.be/Ms3h-eqWk5A

uFR Online (UDP)

For uFR Online most important parameters are 'port_name' which is the IP address of the device on the network, and 'interface' which specifies the type of connection. This is the example of UDP connection, and by clicking on 'Reader Open' it will call ReaderOpenEx() method as:
uFCoder.ReaderOpenEx(1, "192.168.1.8", 85, "");

For more details on uFR Online series device and it's configuration, read more in 'uFR Online - Quick Start Guide' document, that can be found here: https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git
Video on how to open uFR Online can be found here: https://youtu.be/L75l2ReA61c

uFR Series reader via OTG

Simplest method for interacting with a uFR series reader via OTG cable would be to call the ReaderOpen*()* method. This method is called just by clicking on the 'Reader Open' button with 'Dodatne opcije' checkbox unchecked and 'µFR uređaj' selected. By doing that, the following method is called:
uFCoder.ReaderOpen()
This method requires no additional parameters, and relies on FTDI API to establish communication with the device. Video on how to open uFR via OTG can be found here: https://youtu.be/28af6oulv1c

Implementation

This section will describe basic implementation of uFCoder library in Android studio and showcase function calls necessary for getting the token.

Include uFCoder in Android Studio project

Follow the instructions below to include the library in your Android project.

1. Create new or open existing Android project.
2. Open project folder in File Explorer.
3. Navigate to ''<MyProject>/app'' directory.
4. Create a new folder 'libs'.
5. Copy ''uFCoder.aar'' file into ''libs'' folder.

Build Gradle configuration

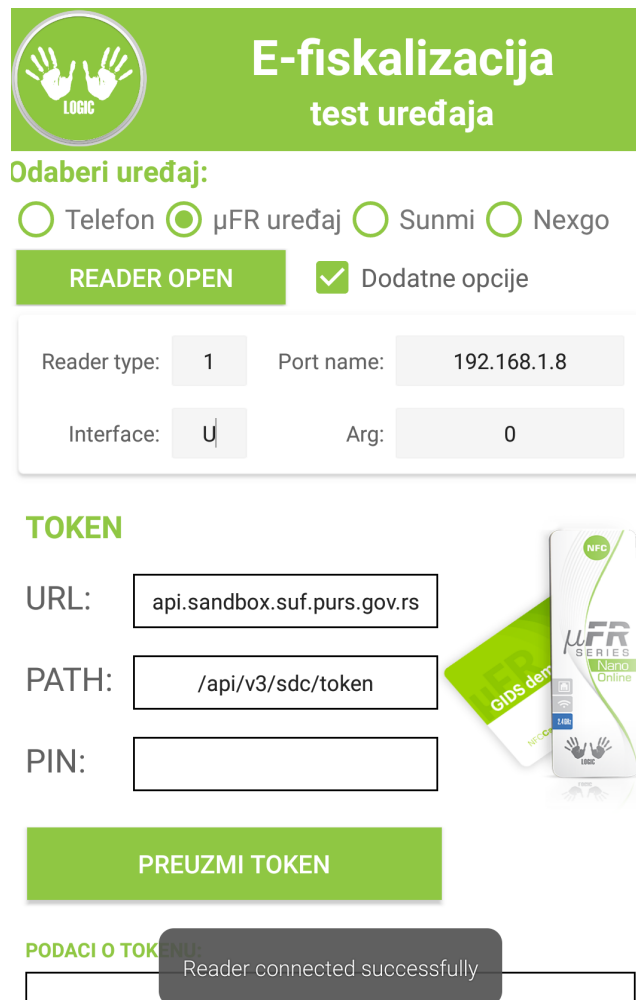Follow the instructions below to configure the build gradle for using the library.

1. Open build.gradle file (the one under 'app')
2. Add **implementation files('libs/uFCoder.aar')** under dependencies.
3. Click on the 'sync now' button to sync projects with Gradle files.

Android example walkthrough

First step, in almost every example in our SDK, would be utilizing ReaderOpen (or ReaderOpenEx which requires providing advanced parameters) to open communication with the device of your choice. For the purpose of this demo, we will use our [uFR Online Series reader](#).

Reader open

- Select 'µFR uređaj' and, depending on device, select 'Dodatne opcije' for providing advanced parameters and using the *ReaderOpenEx()*, instead of the *ReaderOpen()* function by default.
- Fill in the parameters according to the device type. For the purpose of this demo, we'll start by using the uFR Online series reader. Parameters for this use case would be as following:

Image 1. Successful 'Reader Open' with advanced parameters

Digital Logic Ltd.

Code equivalent of calling 'Reader Open' with these parameters would be the following line in Android:
status = uFCoder.ReaderOpenEx(1, '192.168.1.8', 85, "");

At the bottom, a message with the results of our action will appear. Successful 'Reader Open' would display the message as it is on the previous image, otherwise a message with error code will be shown..

Get token

- After a successful 'Reader Open' we can use the rest of the software properly.
- Next option would be to get a token via the 'Preuzmi token' button. This requires valid input of parameters 'URL', 'PATH', and finally 'PIN' which varies depending on your card. Successful result of this action will appear as following:

Image 2. Token results.

-

Code necessary for execution of 'Preuzmi Token' is equivalent to the following snippet in Android:

```
byte null_cert[] = new byte[10];
status[0] = uFCoder.DL_TLS_SetClientCertificate(2, null_cert);
if (status[0] == 0) {
    int[] ret_status = new int[1];
    String resStr = uFCoder.DL_TLS_Request(ret_status, URL, PATH, 443, PIN);
}
```

Function *DL_TLS_SetClientCertificate()* - is used for user certificate authentication with our TLS 1.2 HTTPS client. For more details on the *DL_TLS_SetClientCertificate()* method, refer to 'uFR Series NFC reader API': https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git

Function *DL_TLS_Request* is used to transceive HTTPS GET request over TLS 1.2 secure connection, implementing TLS/SSL user certificate authentication on server request. Request doesn't contain HTTP body and use minimal of the HTTP headers.

For more details on the *DL_TLS_Request ()* method and our TLS support, refer to 'uFR Series NFC reader API': https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git

Custom APDU commands

- Next option is the sending of custom APDU commands, based on sending initially the separate 'Select APDU' via 'Pošalji Select APDU' button, and then APDU commands based on your input via 'Pošalji APDU komandu' button.
- By default, the 'Select' APDU command used in this software has the following value: '*00A4040010A000000748464A492D546178436F726500*'. Successful execution yields the following result:

APDU komande

"SELECT" APDU

00A4040010A000000748464A492D546178436F726500

POŠALJI SELECT APDU

APDU KOMANDA:

88040400000000

POŠALJI APDU KOMANDU

APDU ODGOVOR:

Card detected... Sending "SELECT" command:
[C]: 00A4040010A000000748464A492D546178436F726500
status: [0x00 (0)] UFR_OK
[R]: 9000

O APLIKACIJI

LICENCA

Image 3. 'Select APDU' result

Custom APDU command – Extended APDU

- Finally, we have a demo of an extended APDU command, with the default value of '88040400000000' in the input field. This command, if successful, will yield a large amount of data in one response. The successful execution should look like this:

APDU KOMANDA:

88040400000000

POŠALJI APDU KOMANDU

APDU ODGOVOR:

Card detected... Sending "SELECT" command:
[C]: 00A4040010A000000748464A492D546178436F726500
status: [0x00 (0)] UFR_OK
[R]: 9000

status: [0x00 (0)] UFR_OK
[R]:
308205CE308204B6A003020102020E6220971F98E6F28900000
00004A4300D06092A864886F70D01010D05003049310B30090
603550406130252533117301506035504081306F7265736B
61205570726176613121301F06035504031318536616E64626F7
82053554620497375696E67204341203130301E170D32313039
323130383230303039A170D3234303932313038333030395A30
81C3310B3009060355040613025253311B301906035504080C
12D09FD0BED0B6D0B0D180D0B5D0B2D0B0D18631123010060
35504071309506F7A61726576616331173015060355040919130E

Image 4. Extended APDU command output

F713992B4988557F6137602BE9B846DBA70ACEBA1D2248358
DA318AA522FAAD39E1CC4AEACF09638EB193B956A3D90555C
8FC38B8BA7A816673C9ED379E893B7D0AA4F60D164915F590
7AE201C2078C9D6039F28754EAC7C84FEC187A176619EBA57
0B855F86989D3EDFA85A9915DE2C29C6944CAD17068156E04
CEBDF0B07B2BB8A849680850F1E50C8FFA683C177B29B16CD
741AF0CC37FBBFE53AB048E405390203010001A38202373082
0233300F0603551D130101FF04053003020100300E0603551D0
F0101FF04040302049030170603551D250410300E060C2B0601
0401838620050803033081C50603551D200481BD3081BA3081
B7060C2B0601040183862005080402308A1A6303206082B0601
050507020116266874747703A2F2F706B692E73616E64626F782
E7375662E707572732E676F762E72732F706B69307006082B06
01050507020230641E6200730061006E00640062006F00780005
00073006500200070004B00490053007200620069006A006100
530061006E00640062006F0078002000690073007300750061006
06E00630065050006F006C006900630063007900200020063006C0061
00730073007300323032060B2B060104018386200508050042368747
470733A2F2F6170692E73616E64626F782E7375662E70757572732
E676F762E72733018060B2B060104018386200508060409313
131333835343434301D0603551D0E04160414D3499EE620209
0D32EED5E64EF4D393DFC8C0577301F0603551D23041830168
014332482ABBB7BF155698118B8DE971B0BDFE4A1E1304A060
3551D1F04433041303FA03DA03B8639687474703A2F2F706B6
92E73616E64626F782E7375662E707572732E676F762E72732F
706B692F5355546494943413153616E64626F782E63726C3055060
82B0601050507010104493047304506082B0601050507300286
39687474703A2F2F706B692E73616E64626F782E7375662E707
572732E676F762E72732F706B692F535546494943413153616E64
626F782E636572300D06092A864886F70D01010D0500038201
01008B12C9CCF9A58066FA2C07419800949D9CDE269B5F0CF
CD23EAD93DF796F96BDAC673EA9DD7C9523A4DC3E556BCF6
BCE9CD4021D3368340A5BB9EE8C5072FE30679F7A66052EAC
CA05071FD622A35B62F6D654F4BC9CCAC32F611E905911B0E
61B50BDF9E4E0EE23CA4B71E3B6CC74279F5C11BE7FF073DB
BB1E7DD9B3A84E8DF5976E7006B6A26B8B46905CA3BA0068A
B4F46EEF5A7130AE4B7AADAD56687FB3BB6F1FB168FB03867
B2F61AE5D3264D48704AC748B4E0FC551DFC94B1B3B2318A5
5889D7876CC6137D5380B50CACDA093557A8D2CFDBCBDAB9
8EA2F341108EB359B44A9B40D376F93F5035A4B2F829DC78F3
C0C1B5367083FEB35E73484D03F9000
Received: 1492 bytes

O APLIKACIJI

LICENCA

Image 5. Completed extended APDU command output

Digital Logic Ltd.

Transmission of both 'Select' command and the 'Extended APDU' commands demonstrated can be done in the following manner in Android:

ret_status[0] = uFCoder.APDUHexStrTransceive(send_cmd_string, resp_string);

By using the *APDUHexStrTransceive* function, you can send C–APDU in the c_string (zero terminated) format, containing pairs of the hexadecimal digits.

This software is available on the Play store:
https://play.google.com/store/apps/details?id=com.dlogic.e_fiskaltest

uFCoder library API example - get token minimal snippet

```
package com.dlogic.example;

import com.dlogic.uFCoder;

public class MainActivity extends AppCompatActivity {

  static {
      System.loadLibrary("uFCoder"); //Load uFCoder library
  }

   int status;
   uFCoder uFCoder; //Create uFCoder class instance

  @Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);

     uFCoder = new uFCoder(getApplicationContext(), this);

     status = uFCoder.ReaderOpen();
    // if you wish to use the internal Android NFC reader and/or use other supported devices, implement
    // ReaderOpenEx with parameters you need. e.g Android internal NFC reader parameters
    // status = uFCoder.ReaderOpenEx(5, "", 0, "");
    if(status == 0x00) // if the device open was successful
    {
        byte null_cert[] = new byte[10];
         status = uFCoder.DL_TLS_SetClientCertificate(2, null_cert);
```

```
        if (status == 0) {
            String tokenResult = uFCoder.DL_TLS_Request(status, URL, PATH, 443, PIN);
            if (status == 0) {
                // if the status was a valid one, handle result here
            } else {
                // otherwise handle errors in this segment
            }
        }
    }
}
```

Full code of this snippet, as an Android example, can be found in our SDK git repository:
https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-android-efiscalisation_test.git

uFCoder library API example - send APDU minimal snippet

```
package com.dlogic.example;

import com.dlogic.uFCoder;

public class MainActivity extends AppCompatActivity {

    static {
        System.loadLibrary("uFCoder"); // Load uFCoder library
    }

    int status;
    uFCoder uFCoder; // Create uFCoder class instance

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        uFCoder = new uFCoder(getApplicationContext(), this);

        int status = 0x01;
        status = uFCoder.ReaderOpen();
        // if you wish to use the internal Android NFC reader and/or use other supported devices, implement
        // ReaderOpenEx with parameters you need. e.g Android internal NFC reader parameters
```

```
//    status = uFCoder.ReaderOpenEx(5, "", 0, "");
      if(status == 0x00) // if the device open was successful
      {
            status = uFCoder.SetISO14443_4_Mode(); // this function is mandatory for uFR devices, omit if
                                                    //  you're not using uFR Series reader
            if (status == 0) {
              resp = uFCoder.APDUHexStrTransceive(ret_status, ApduCmd_Str);
              //alternatively you can use uFCoder.APDUPlainTransceive() method for sending an APDU
              // command as a byte array instead of a hex string
              if (status == 0) {
                  // if the status was a valid one, handle result 'resp' here
              } else {
                  // otherwise handle errors in this segment
              }
              uFCoder.s_block_deselect((byte) 100); // mandatory so that uFR reader can resume polling
                                                    // use this method when you're finished with
                                                    //sending/receiving of APDU commands.
            }
        }
      }
}
```

For more details on our API and function calls, refer to 'uFR Series NFC reader API':
https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git

References

Google Play

uFR e-Fiscalisation app: https://play.google.com/store/apps/details?id=com.dlogic.e_fiskaltest
uFR GIDS demo: https://play.google.com/store/apps/details?id=com.dlogic.ufrgidsdemo

Youtube

D-Logic NFC SDK channel: https://www.youtube.com/channel/UCNYYw0iUoua3bMTJbSuaWQA
uFR e-Fiscalisation demo with Nexgo reader: https://youtu.be/34rpoPHeHCA
uFR e-Fiscalisation demo with Sunmi V2 reader: https://youtu.be/Ms3h-eqWk5A
uFR GIDS demo with uFR reader (WiFi connection): https://youtu.be/L75l2ReA61c
uFR GIDS demo with uFR reader (USB connection): https://youtu.be/28af6oulv1c
uFR GIDS demo with Android internal NFC: https://youtu.be/b2QofrpRwHo

SDK examples

uFR e-Fiscalisation - Android source code:

https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-android-efiscalisation_test.git

uFR GIDS demo - Android source code:

https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-android-gids.git

uFR GIDS demo - iOS source code:

https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-ios-gids.git

Documentation

All documents related to our SDK can be found here:

https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git

Important documents related to this manual:
- uFR Series NFC reader API
- uFR Online Quick Start Guide
- using uFR Android library
- using uFR library in XCode

Revision history

Date	Version	Comment
2019-10-31	1.0	Base document